Week 4, Part 1:
Entropy and Algorithms

Prof. Michael Gastpar

Slides by Prof. M. Gastpar and Prof. em. B. Rimoldi

EPFL

Spring Semester 2025

Outline

In today's lecture, we explore the role of entropy in algorithms beyond data compression.

Specifically, we will briefly look at the following examples:

- "20 Questions Problem"
- Sorting
- "Billiard Balls" Puzzle


TWENTY QUESTIONS
DICK
HERB
VAN
FLORENCE
ALDO RAY

UP NEXT

USE ENTROPY

TO TACKLE THIS

PROBLEM.

LAST WEEK

$$H_D(X) = H_D(P) = -\sum_x p(x) \log_D p(x)$$

$$0 \leq H_D(X) \leq \log_D |\mathcal{A}|$$

$$H(X|Y=y) = -\sum_x p(x|y) \log_D p(x|y)$$

$$\hookrightarrow H(X|Y) = \sum_y p(y)\, H(X|Y=y)$$

$$H(X|Y) \leq H(X)$$ WITH EQ IFF X INDEP Y

$$H(X|Y,Z) \leq H(X|Z) \leq H(X)$$

$$H(S_1, S_2, S_3, S_4)$$

$$= H(S_2, S_4, S_1, S_3)$$

$$= H(S_1) + H(S_2|S_1) + H(S_3|S_1,S_2) + H(S_4|S_1,S_2,S_3)$$

$$H(S_1, S_2, S_3, S_4)$$

$$= H(S_2, S_4, S_1, S_3)$$

$$= H(S_1) + H(S_2|S_1) + H(S_3|S_1, S_2) + H(S_4|S_1, S_2, S_3)$$

$$\leq H(S_1) + H(S_2) + H(S_3) + H(S_4)$$

WITH EQ IFF S_1, S_2, S_3, S_4 ARE INDEPENDENT

THE 20 QUESTIONS PROBLEM

Let X be a random variable.

What is the minimum number of "Yes/No questions" needed to identify X?

And which questions should be asked?

SOLUTION

- Consider a binary code Γ for $X \in \mathcal{X}$.
- Once Γ is fixed, we know $x \in \mathcal{X}$ iff we know the codeword $\Gamma(x)$.
- The strategy consists in asking the ith question so as to obtain the ith bit of the codeword $\Gamma(x)$.
- The average number of questions is $L(X, \Gamma)$, which is minimized if Γ is the encoding map of a Huffman code.
- We will see that we cannot do better.

First an example.

SOLUTION

uniquely decodable

- Consider a binary code Γ for $X \in \mathcal{X}$.
- Once Γ is fixed, we know $x \in \mathcal{X}$ iff we know the codeword $\Gamma(x)$.
- The strategy consists in asking the ith question so as to obtain the ith bit of the codeword $\Gamma(x)$.
- The average number of questions is $L(X, \Gamma)$, which is minimized if Γ is the encoding map of a Huffman code.
- We will see that we cannot do better.

First an example.

SOLUTION

- Consider a binary code Γ for $X \in \mathcal{X}$.
- Once Γ is fixed, we know $x \in \mathcal{X}$ iff we know the codeword $\Gamma(x)$.
- The strategy consists in asking the ith question so as to obtain the ith bit of the codeword $\Gamma(x)$.
- The average number of questions is $L(X, \Gamma)$, which is minimized if Γ is the encoding map of a Huffman code.
- We will see that we cannot do better.

First an example.

SOLUTION

- Consider a binary code Γ for $X \in \mathcal{X}$.
- Once Γ is fixed, we know $x \in \mathcal{X}$ iff we know the codeword $\Gamma(x)$.
- The strategy consists in asking the ith question so as to obtain the ith bit of the codeword $\Gamma(x)$.
- The average number of questions is $L(X, \Gamma)$, which is minimized if Γ is the encoding map of a Huffman code.
- We will see that we cannot do better.

First an example.

SOLUTION

- Consider a binary code Γ for $X \in \mathcal{X}$.
- Once Γ is fixed, we know $x \in \mathcal{X}$ iff we know the codeword $\Gamma(x)$.
- The strategy consists in asking the ith question so as to obtain the ith bit of the codeword $\Gamma(x)$.
- The average number of questions is $L(X, \Gamma)$, which is minimized if Γ is the encoding map of a Huffman code.
- We will see that we cannot do better.

First an example.

Ex: $\mathcal{X} = \{ \text{cat}, \text{dog} \}$

$p(\text{cat}) = 2/3$

$p(\text{dog}) = 1/3$

Ex: $\mathcal{X} = \{\text{cat}, \text{dog}, \text{pony}\}$

$p(\text{cat}) = 1/2$

$p(\text{dog}) = 1/4$

$p(\text{pony}) = 1/4$

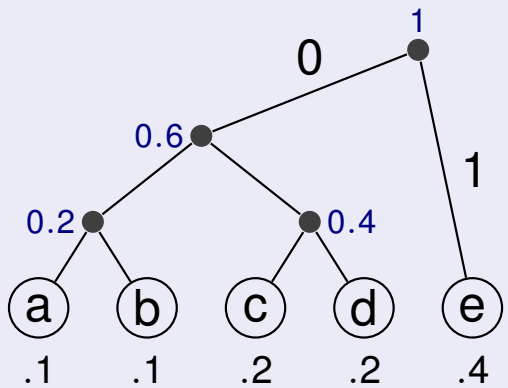
EXAMPLE

Let X be a random variable in $\mathcal{A} = \{a, b, c, d, e\}$ having distribution p_X:

X	a	b	c	d	e
p_X	0.1	0.1	0.2	0.2	0.4

Example (cont.)

We construct a binary Huffman code for X:



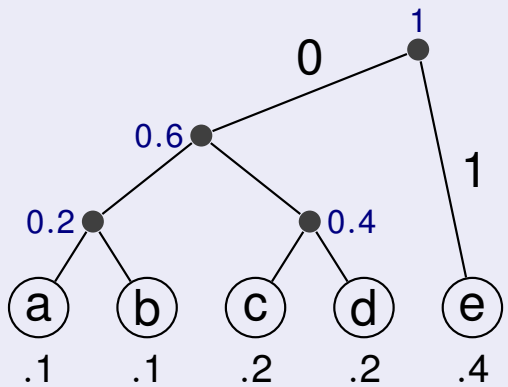
$\mathcal{A}$	Γ_H
a	000
b	001
c	010
d	011
e	1

Suppose that the realization is $X = b$ (but we do not know it).

Example (cont.)

The strategy is to identify b via its binary codeword.

With the first question we try to find the first letter of the codeword:



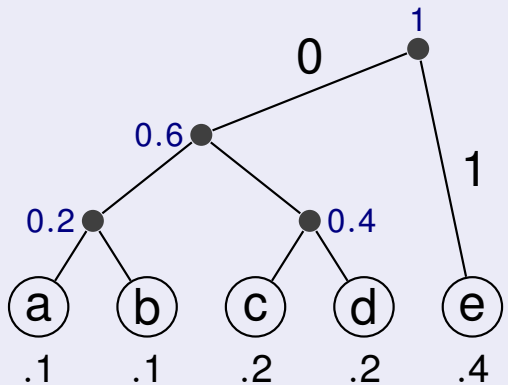
$\mathcal{A}$	Γ_H
a	000
b	001
c	010
d	011
e	1

We ask the question: Is $X \in \{e\}$?

The answer is NO. We know that the first letter of the codeword is 0.

Example (cont.)

With the second question we find the second codeword letter:



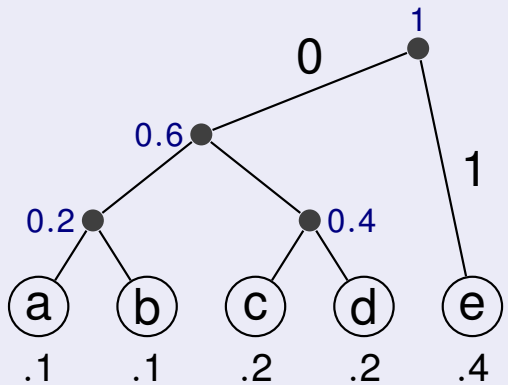
$\mathcal{A}$	Γ_H
a	000
b	001
c	010
d	011
e	1

We ask the question: Is $X \in \{c, d\}$?

The answer is NO. We know that the second letter of the codeword is 0.

Example (cont.)

With the third question we find the third codeword letter:



$\mathcal{A}$	Γ_H
a	000
b	001
c	010
d	011
e	1

We ask the question: is $X = b$?

The answer is Yes. We know that the third letter of the codeword is 1 and that $X = b$.

Optimality of the Huffman querying strategy

We have seen that a prefix-free code for $X \in \mathcal{X}$ leads to a querying strategy to find the realization of X.

Similarly, a deterministic querying strategy leads to a binary prefix-free code for X. Here is why:

- Before the first question we know that $x \in \mathcal{X}$.
- Without loss of generality, the first question can be formulated in terms of "Is $x \in \mathcal{A}$?" for some $\mathcal{A} \subset \mathcal{X}$. (The choice of $\mathcal{A}$ is determined from the strategy, that we fix once and for all.)
- If the answer is YES, then we know that $x \in \mathcal{A} \subset \mathcal{X}$. Otherwise $x \in \mathcal{A}^c \subset \mathcal{X}$. Either way we have reduced the size of the set that contains x.
- We continue asking similar questions until the value of x is fully determined, then we stop.

Optimality of the Huffman querying strategy

We have seen that a prefix-free code for $X \in \mathcal{X}$ leads to a querying strategy to find the realization of X.

Similarly, a deterministic querying strategy leads to a binary prefix-free code for X. Here is why:

- Before the first question we know that $x \in \mathcal{X}$.
- Without loss of generality, the first question can be formulated in terms of "Is $x \in \mathcal{A}$?" for some $\mathcal{A} \subset \mathcal{X}$. (The choice of $\mathcal{A}$ is determined from the strategy, that we fix once and for all.)
- If the answer is YES, then we know that $x \in \mathcal{A} \subset \mathcal{X}$. Otherwise $x \in \mathcal{A}^c \subset \mathcal{X}$. Either way we have reduced the size of the set that contains x.
- We continue asking similar questions until the value of x is fully determined, then we stop.

Optimality of the Huffman querying strategy

We have seen that a prefix-free code for $X \in \mathcal{X}$ leads to a querying strategy to find the realization of X.

Similarly, a deterministic querying strategy leads to a binary prefix-free code for X. Here is why:

- Before the first question we know that $x \in \mathcal{X}$.
- Without loss of generality, the first question can be formulated in terms of "Is $x \in \mathcal{A}$?" for some $\mathcal{A} \subset \mathcal{X}$. (The choice of $\mathcal{A}$ is determined from the strategy, that we fix once and for all.)
- If the answer is YES, then we know that $x \in \mathcal{A} \subset \mathcal{X}$. Otherwise $x \in \mathcal{A}^c \subset \mathcal{X}$. Either way we have reduced the size of the set that contains x.
- We continue asking similar questions until the value of x is fully determined, then we stop.

Optimality of the Huffman querying strategy

We have seen that a prefix-free code for $X \in \mathcal{X}$ leads to a querying strategy to find the realization of X.

Similarly, a deterministic querying strategy leads to a binary prefix-free code for X. Here is why:

- Before the first question we know that $x \in \mathcal{X}$.
- Without loss of generality, the first question can be formulated in terms of "Is $x \in \mathcal{A}$?" for some $\mathcal{A} \subset \mathcal{X}$. (The choice of $\mathcal{A}$ is determined from the strategy, that we fix once and for all.)
- If the answer is YES, then we know that $x \in \mathcal{A} \subset \mathcal{X}$. Otherwise $x \in \mathcal{A}^c \subset \mathcal{X}$. Either way we have reduced the size of the set that contains x.
- We continue asking similar questions until the value of x is fully determined, then we stop.

Optimality of the Huffman querying strategy

We have seen that a prefix-free code for $X \in \mathcal{X}$ leads to a querying strategy to find the realization of X.

Similarly, a deterministic querying strategy leads to a binary prefix-free code for X. Here is why:

- Before the first question we know that $x \in \mathcal{X}$.
- Without loss of generality, the first question can be formulated in terms of "Is $x \in \mathcal{A}$?" for some $\mathcal{A} \subset \mathcal{X}$. (The choice of $\mathcal{A}$ is determined from the strategy, that we fix once and for all.)
- If the answer is YES, then we know that $x \in \mathcal{A} \subset \mathcal{X}$. Otherwise $x \in \mathcal{A}^c \subset \mathcal{X}$. Either way we have reduced the size of the set that contains x.
- We continue asking similar questions until the value of x is fully determined, then we stop.

Optimality of the Huffman querying strategy

We have seen that a prefix-free code for $X \in \mathcal{X}$ leads to a querying strategy to find the realization of X.

Similarly, a deterministic querying strategy leads to a binary prefix-free code for X. Here is why:

- Before the first question we know that $x \in \mathcal{X}$.
- Without loss of generality, the first question can be formulated in terms of "Is $x \in \mathcal{A}$?" for some $\mathcal{A} \subset \mathcal{X}$. (The choice of $\mathcal{A}$ is determined from the strategy, that we fix once and for all.)
- If the answer is YES, then we know that $x \in \mathcal{A} \subset \mathcal{X}$. Otherwise $x \in \mathcal{A}^c \subset \mathcal{X}$. Either way we have reduced the size of the set that contains x.
- We continue asking similar questions until the value of x is fully determined, then we stop.

- The sequence of YES/NO answers is a binary codeword associated to x.
- The code obtained when we consider all possible values of x is a binary prefix-free code.
- Since the tree is prefix-free, its average codeword-length cannot be smaller than that of a Huffman code.

- The sequence of YES/NO answers is a binary codeword associated to x.
- The code obtained when we consider all possible values of x is a binary prefix-free code.
- Since the tree is prefix-free, its average codeword-length cannot be smaller than that of a Huffman code.

- The sequence of YES/NO answers is a binary codeword associated to x.
- The code obtained when we consider all possible values of x is a binary prefix-free code.
- Since the tree is prefix-free, its average codeword-length cannot be smaller than that of a Huffman code.

SORTING VIA PAIRWISE COMPARISONS

GIVEN AN UNSORTED LIST WITH n ELEMENTS:

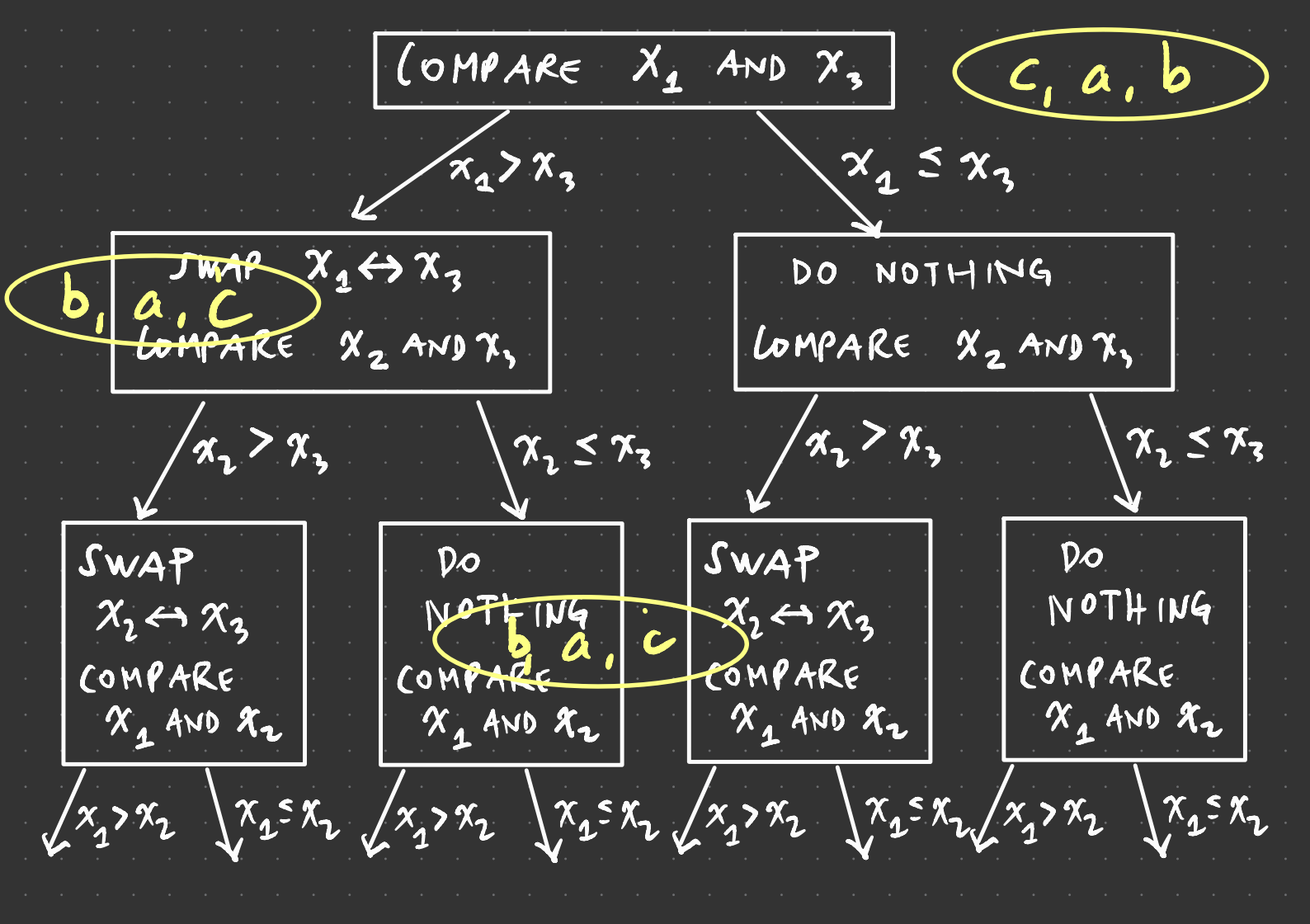
c, a, b ($n = 3$)

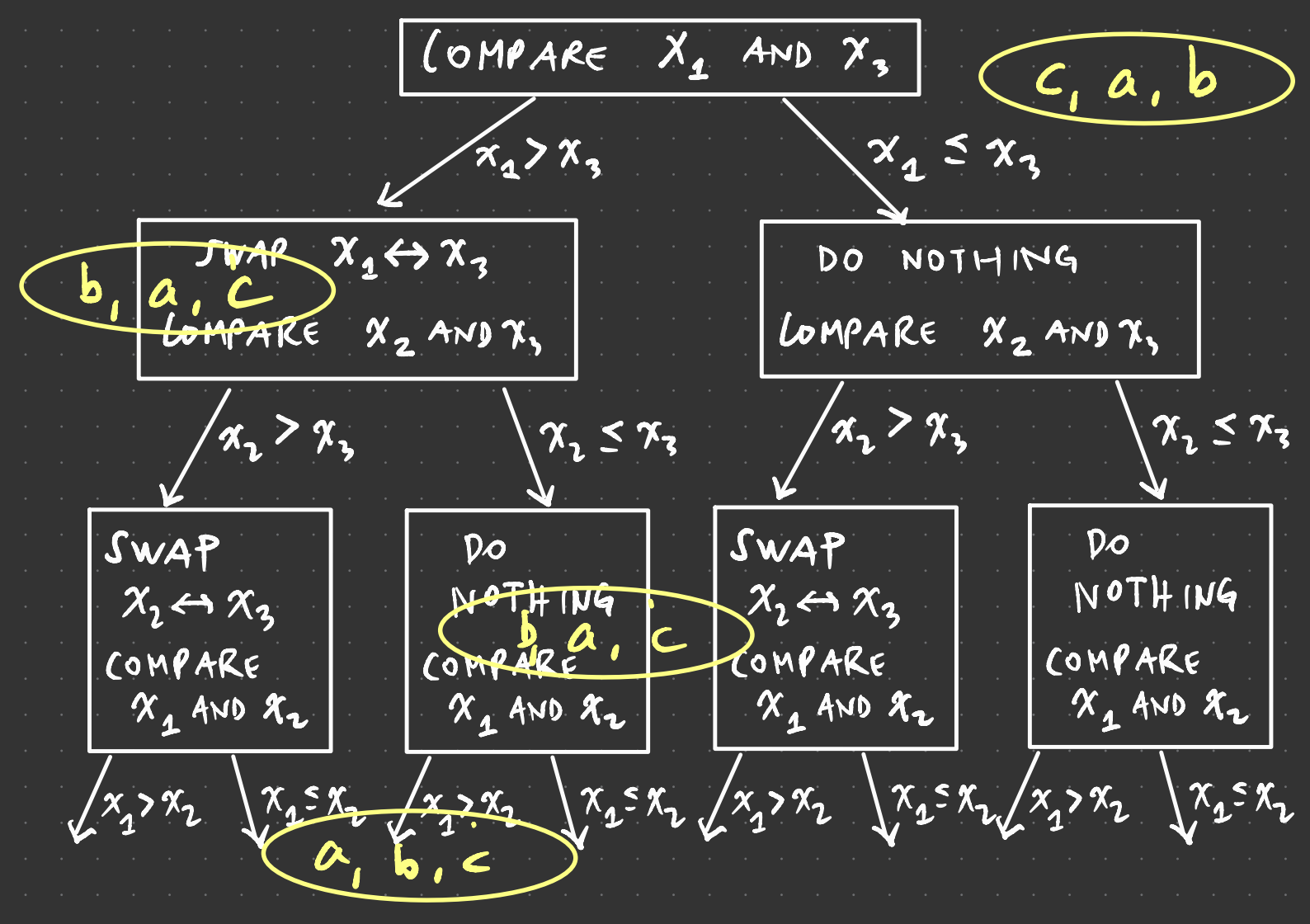
Repeat:

1) select two positions
$1 \leq i < j \leq n$

2) "compare and swap":

If $x_i > x_j$
then swap elements $x_i \leftrightarrow x_j$
Else do nothing


COMPARE x_1 AND x_3
c, a, b
$x_1 > x_3$
$x_1 \le x_3$
SWAP $x_1 \leftrightarrow x_3$
COMPARE x_2 AND x_3
b, a, c
DO NOTHING
COMPARE x_2 AND x_3
$x_2 > x_3$
$x_2 \le x_3$
$x_2 > x_3$
$x_2 \le x_3$
SWAP $x_2 \leftrightarrow x_3$
COMPARE x_1 AND x_2
DO NOTHING
COMPARE x_1 AND x_2
b, a, c
SWAP $x_2 \leftrightarrow x_3$
COMPARE x_1 AND x_2
DO NOTHING
COMPARE x_1 AND x_2
$x_1 > x_2$
$x_1 \le x_2$
$x_1 > x_2$
$x_1 \le x_2$
$x_1 > x_2$
$x_1 \le x_2$
$x_1 > x_2$
$x_1 \le x_2$


COMPARE x_1 AND x_3
c, a, b
$x_1 > x_3$
$x_1 \leq x_3$
SWAP $x_1 \leftrightarrow x_3$
COMPARE x_2 AND x_3
b, a, c
DO NOTHING
COMPARE x_2 AND x_3
$x_2 > x_3$
$x_2 \leq x_3$
$x_2 > x_3$
$x_2 \leq x_3$
SWAP $x_2 \leftrightarrow x_3$
COMPARE x_1 AND x_2
DO NOTHING
COMPARE x_1 AND x_2
b, a, c
SWAP $x_2 \leftrightarrow x_3$
COMPARE x_1 AND x_2
DO NOTHING
COMPARE x_1 AND x_2
$x_1 > x_2$
$x_1 \leq x_2$
$x_1 > x_2$
$x_1 \leq x_2$
a, b, c
$x_1 > x_2$
$x_1 \leq x_2$
$x_1 > x_2$
$x_1 \leq x_2$

OBSERVATION 1

THE SEQUENCE OF PAIRWISE COMPARISONS MUST IDENTIFY THE EXACT ORDER OF THE UNSORTED LIST.

OBSERVATION 2

THE SEQUENCE OF PAIRWISE COMPARISONS IS A UNIQUELY DECODABLE (ACTUALLY, PREFIX-FREE) BINARY CODE FOR X.

Therefore, we must have:

$$\mathbb{E}[\text{number of comparisons}] \geq H_2(X)$$

WHAT IS X ?

1) WHAT ALPHABET DOES IT LIVE IN?

$n=3$: $\mathcal{X} = \{ abc, acb, bac, bca, cab, cba \}$

$\mathcal{X}$: set of all permutations.

$|\mathcal{X}| = n!$

2) WHAT IS $p(x)$?

OUR ALGORITHM SHOULD WORK FOR ALL $p(x)$.

Therefore, we must have:

$$\mathbb{E}[\text{number of comparisons}] \geq \max_{p(x)} H_2(X) = \log_2 |X| = \log_2 n!$$

Known Bounds on Factorial

$$\frac{n^n}{e^{n-1}} \leq n! \leq \frac{n^{n+1}}{e^{n-1}}$$

uniform X

$$\hookrightarrow H_2(X) \approx \log_2 \frac{n^n}{e^{n-1}}$$

$$= n \log_2 n - (n-1) \log_2 e$$

- Sorting by pairwise comparisons with binary output.
  That is, for each comparison, the answer is either " $<$ " or " $\geq$ ".

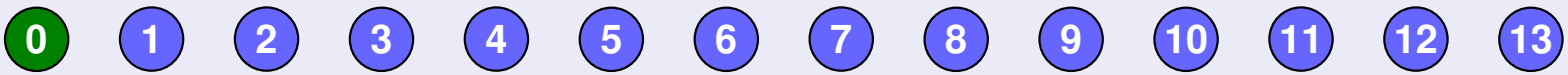
- Suppose we have an unordered list of n objects that need to be sorted.

- How many binary comparisons are needed?

- Let us tackle this question via entropy.

EXERCISE

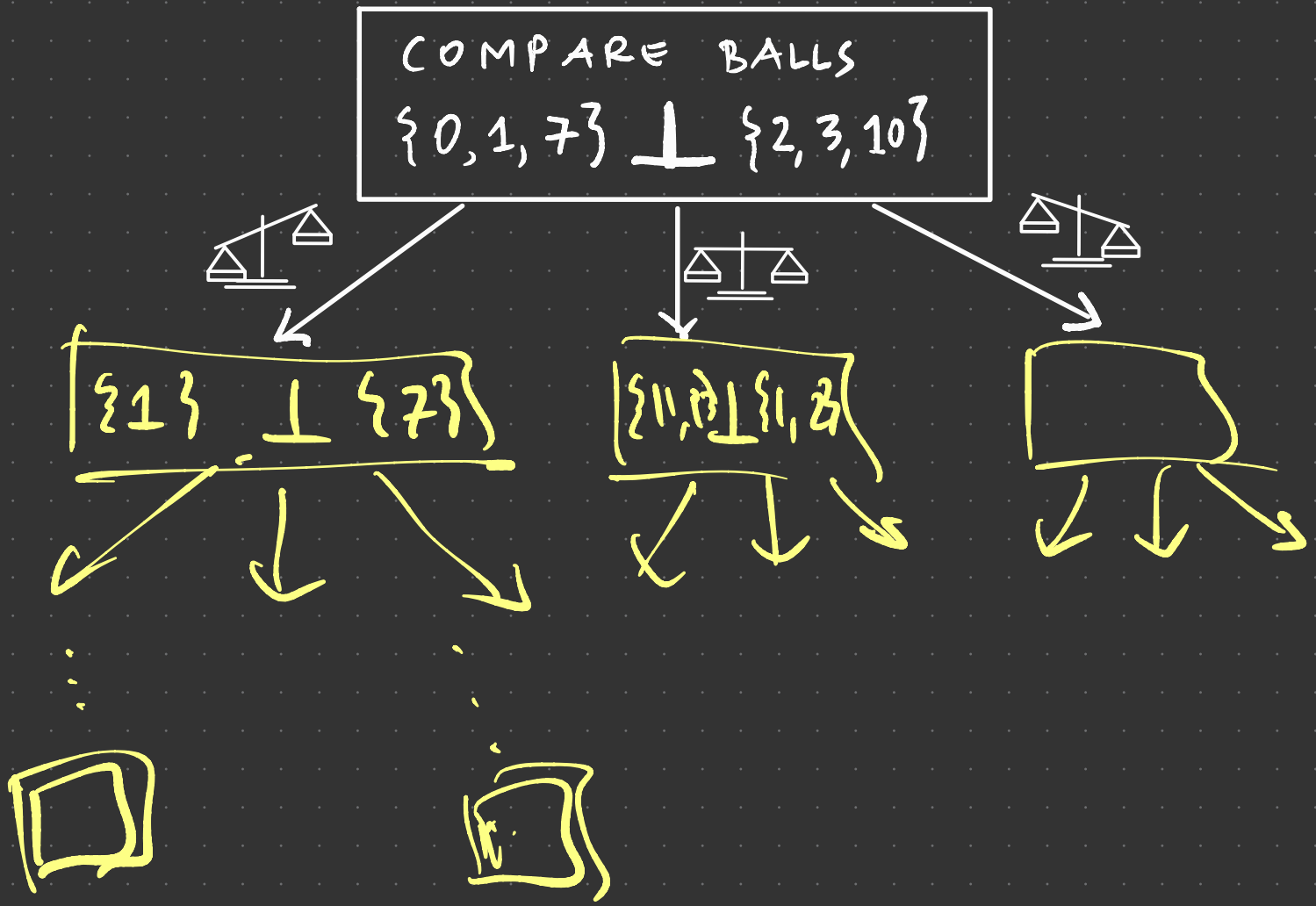
There are 14 billiard balls numbered as shown:



Among balls 1 - 13, at most one **could** be heavier/lighter than the others.

What is the minimum number of weightings to simultaneously determine:

- if one ball is different . . .
- if there is such a ball, which one, . . .
- and whether the different ball is heavier/lighter.


COMPARE BALLS
{0, 1, 7} ⊥ {2, 3, 10}
{1} ⊥ {7}
{11, 2} ⊥ {1, 2}

OBSERVATION 1:

WE ARE SPECIFYING A TERNARY CODE.

OBSERVATION 1:

WE ARE SPECIFYING A TERNARY CODE.

↳ BUT A CODE FOR WHAT ?

A CODE FOR X:

$X = 0$: all balls equal

$X = +1$: ball 1 is heavier

⋮

$X = +13$: ball 13 is heavier

$X = -1$: ball 1 is lighter

⋮

$X = -13$: ball 13 is lighter

$|\mathcal{X}| = 27$

COMPARE BALLS

$\{0, 1, 7\} \perp \{2, 3, 10\}$

OBSERVATION:

NUMBER OF WEIGHINGS

$=$

LENGTH OF TERNARY CODEWORD

Theorem:

$$\mathbb{E}[\text{Number of Weighings}] \geq H_3(X)$$

Moreover, our strategy must work *irrespective* of the probability distribution of X.

Theorem:

$$\mathbb{E}[\text{Number of weighings}] \geq \max_{p(x)} H_3(X) = \log_3 27 = 3$$

BUT DOES THERE INDEED EXIST SUCH A CODE (I.E. A WEIGHING STRATEGY)?

FACT: ENTROPY DOES NOT GUARANTEE THE EXISTENCE OF SUCH A STRATEGY !

BUT DOES THERE INDEED EXIST SUCH A CODE (I.E. A WEIGHING STRATEGY)?

↳ LET US SUPPOSE IT EXISTS!

THEN ENTROPY TELLS US A FEW BASIC FACTS.

FACT 1: IF 3 WEIGHINGS S_1, S_2, S_3 UNIQUELY SPECIFY X, THEN WE MUST HAVE

$$H_3(X) = H_3(S_1, S_2, S_3).$$

PROOF:

$$H(X, S_1, S_2, S_3) = H(X) + H(S_1, S_2, S_3 | X) \quad (H(S_1, S_2, S_3 | X) = 0)$$

$$= H(S_1, S_2, S_3) + H(X | S_1, S_2, S_3) \quad (H(X | S_1, S_2, S_3) = 0)$$

PROOF:

$$H_3(X, S_1, S_2, S_3) = 0$$

$$= H_3(X \mid S_1, S_2, S_3) + H_3(S_1, S_2, S_3)$$

$$= H_3(X) + \underbrace{H_3(S_1, S_2, S_3 \mid X)}_{=0}$$

□

LET X BE UNIFORMLY DISTRIBUTED.

FACT 2: IF 3 WEIGHINGS S_1, S_2, S_3 UNIQUELY SPECIFY X, THEN WE MUST HAVE

(a) S_1, S_2, S_3 UNIFORMLY DISTRIBUTED

(b) S_1, S_2, S_3 INDEPENDENT

PROOF:

$$H_3(S_1, S_2, S_3) \leq 3$$

MUST BE

BUT ALSO:

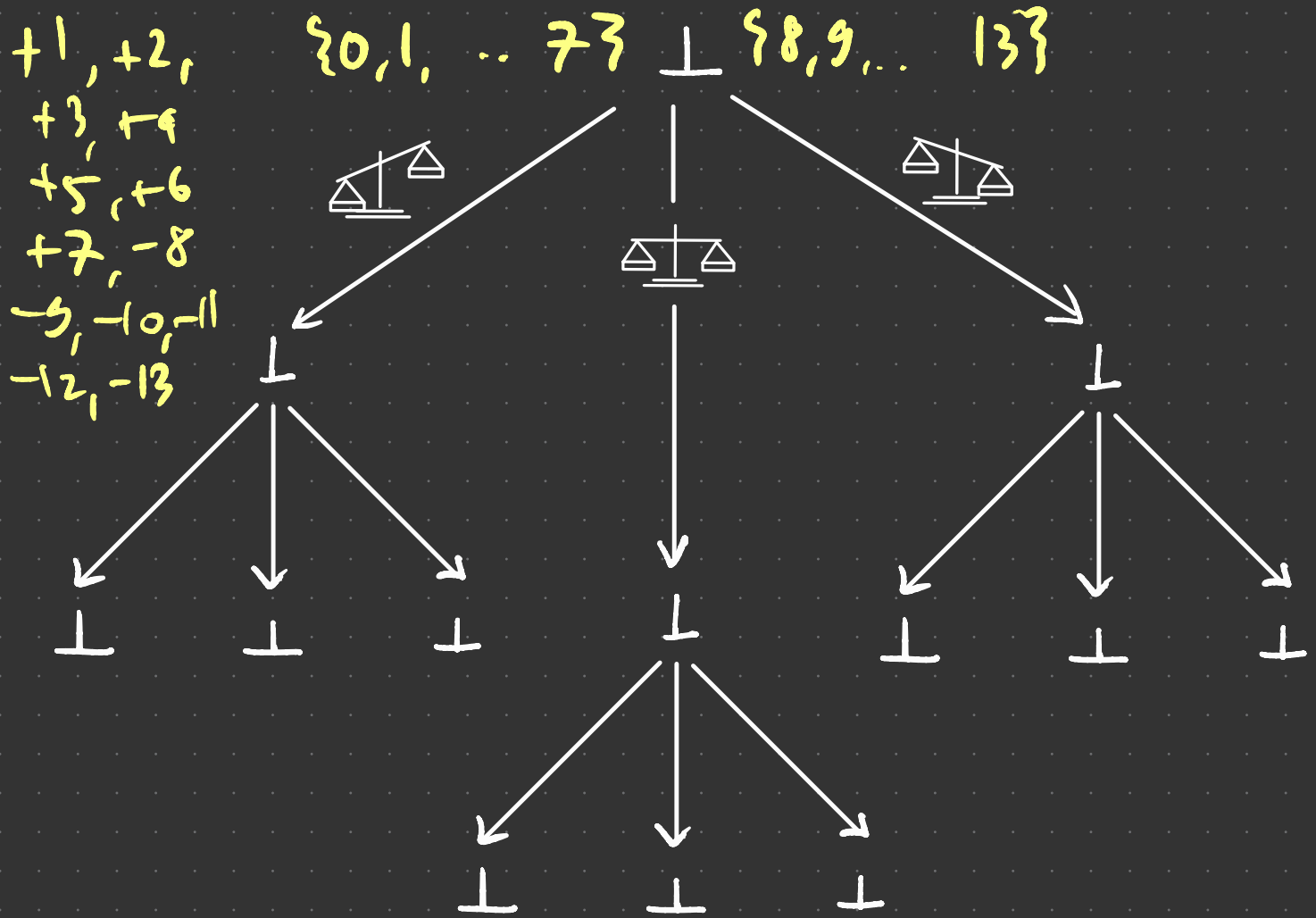
$$H_3(S_1) + H(S_2|S_1) + H(S_3|S_1, S_2)$$

$$\leq H_3(S_1) + H_3(S_2) + H_3(S_3)$$

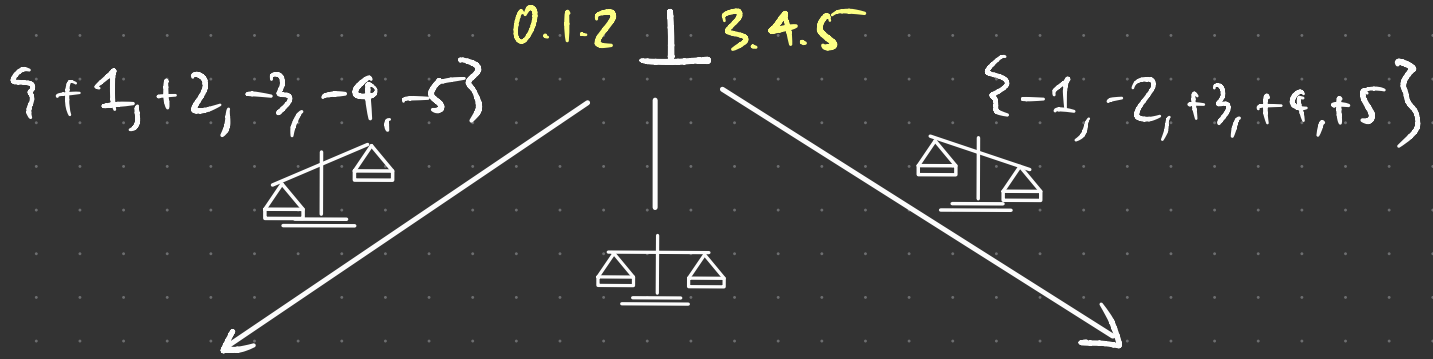
EQ. IFF $S_1 \perp S_2 \perp S_3$

$$\leq \log_3 3 + \log_3 3 + \log_3 3 = 3$$

EQ. IFF UNIFORM.


+1, +2,
+3, +4
+5, +6
+7, -8
-9, -10, -11
-12, -13
{0, 1, .. 7}
⊥
{8, 9, .. 13}

$\{0, +1, -1, +2, -2, \ldots, +13, -13\}$ UNIFORM



HENCE, HERE:

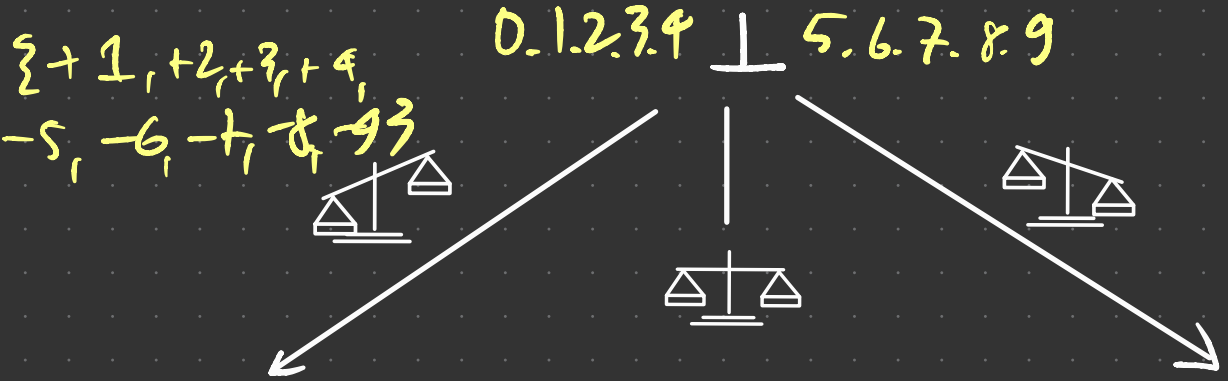
$p(S_1 = \text{⚖}) = 5/27$

$p(S_1 = \text{⚖}) = 17/27$

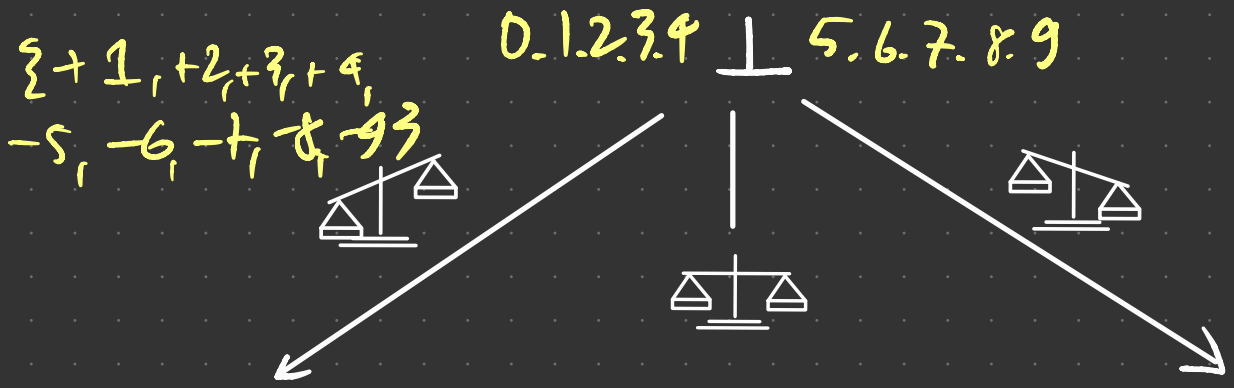
$p(S_1 = \text{⚖}) = 5/27$

} NOT UNIFORM. CANNOT BE OPTIMAL!

$\{0, +1, -1, +2, -2, \ldots, +13, -13\}$ UNIFORM

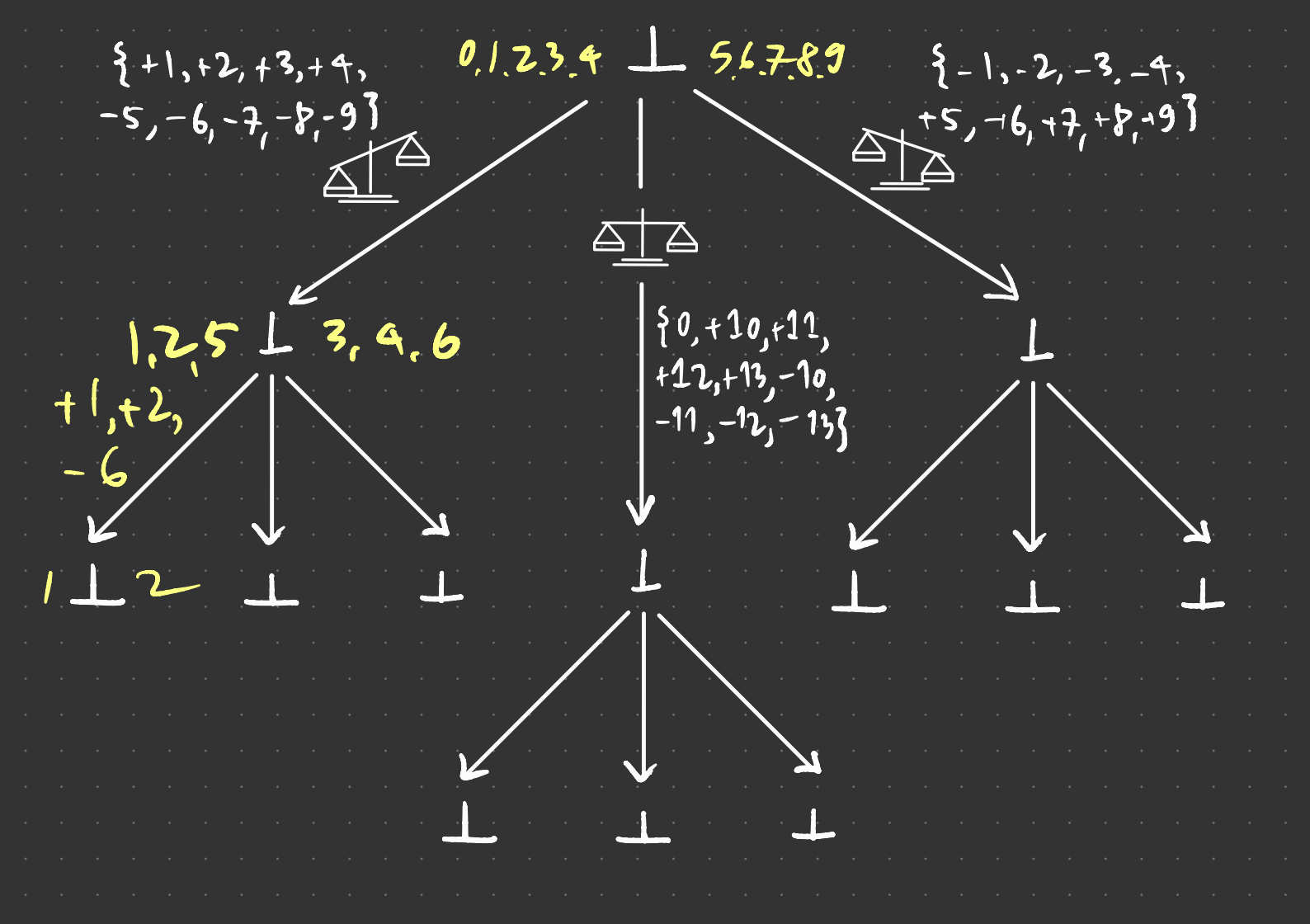


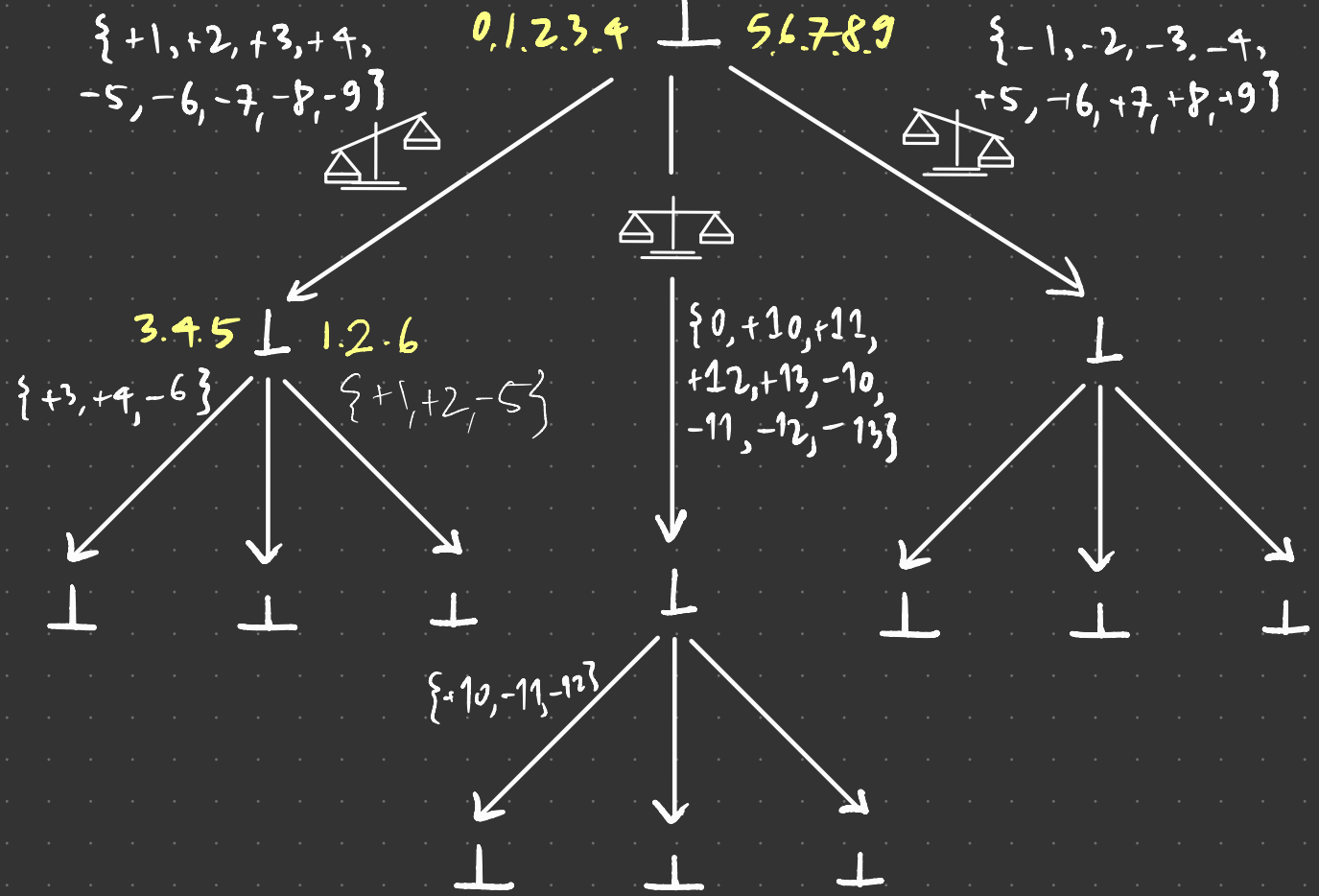
$\{0, +1, -1, +2, -2, \ldots, +13, -13\}$ UNIFORM

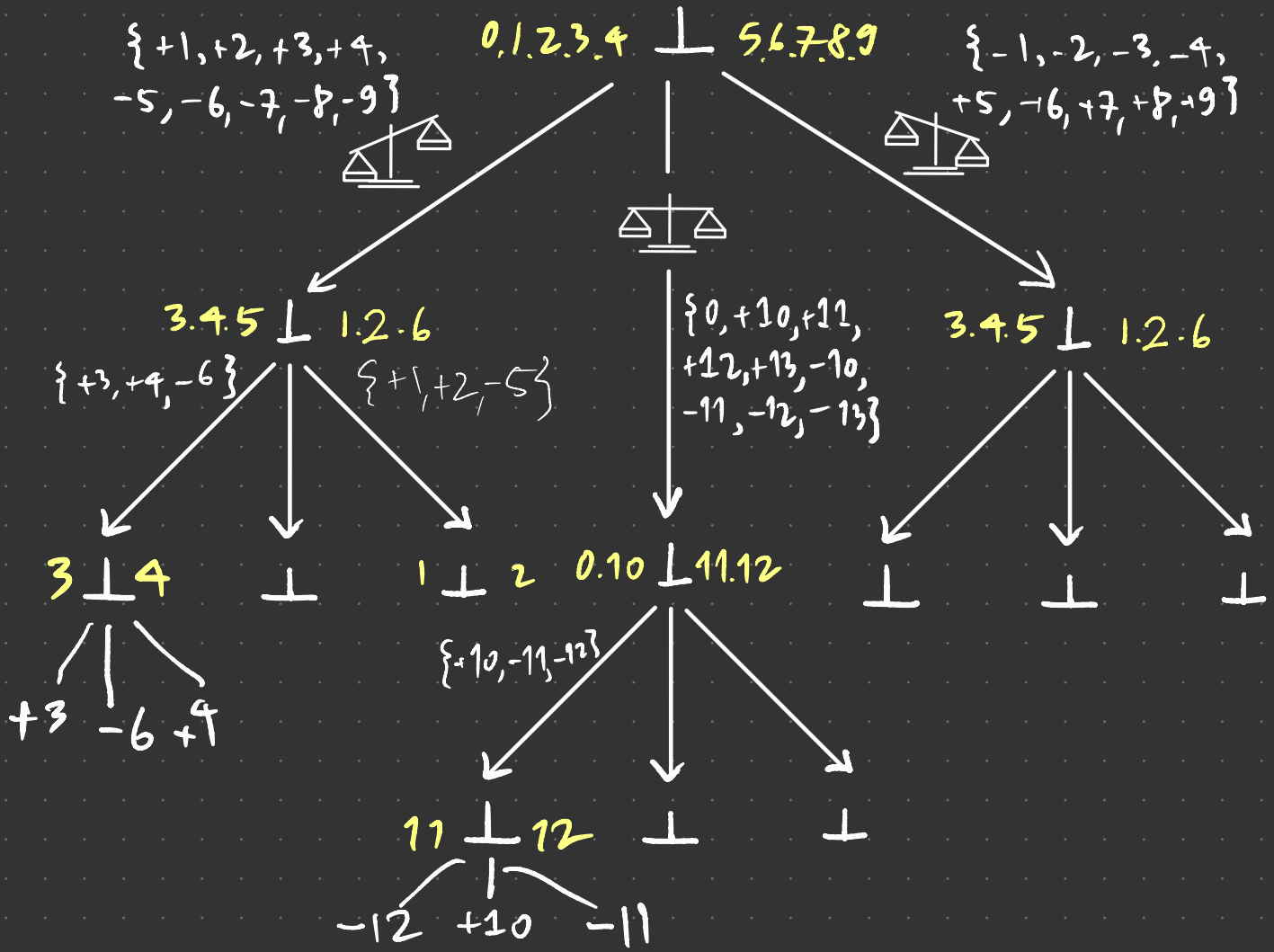


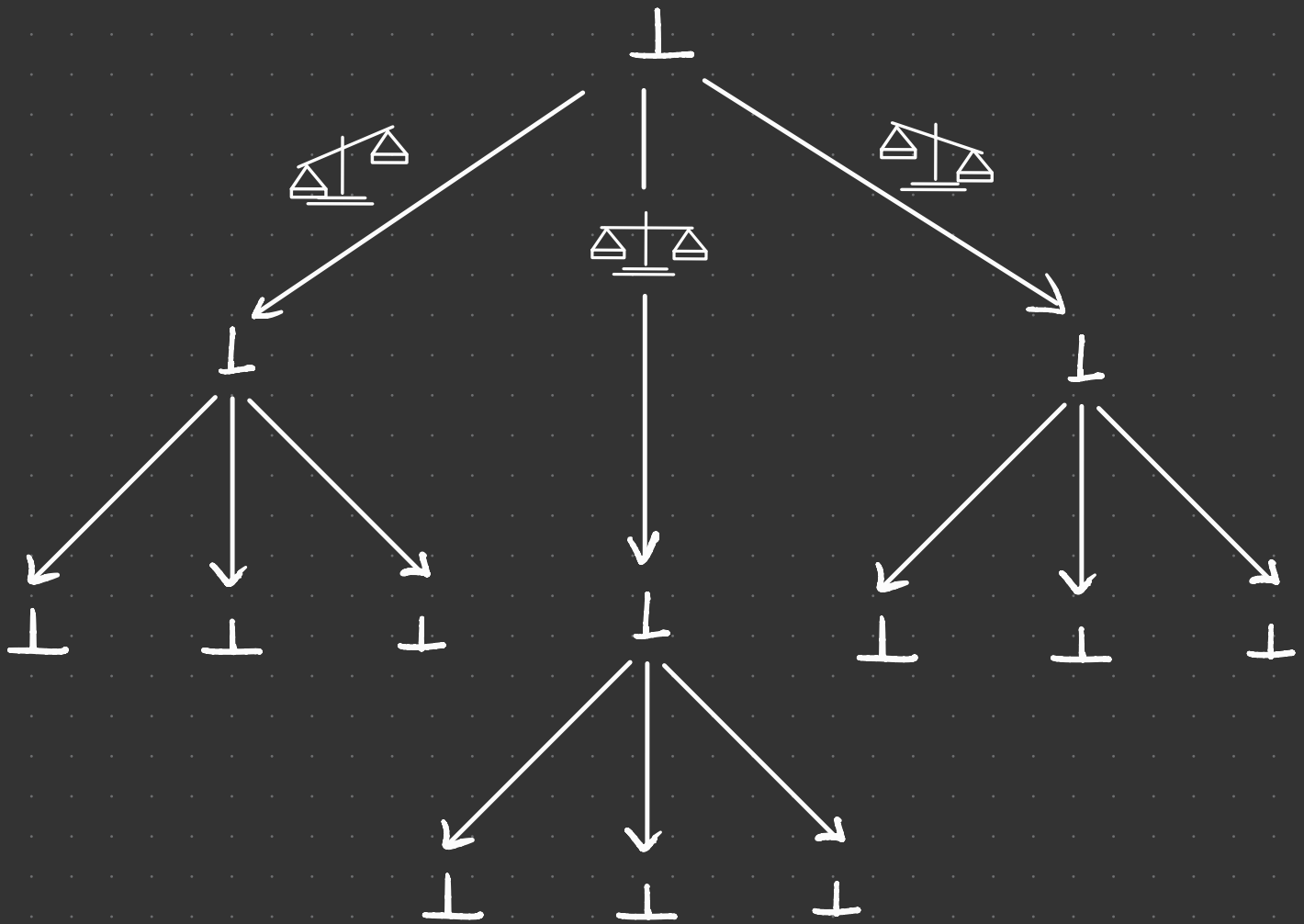
NOW S_1 IS UNIFORM!

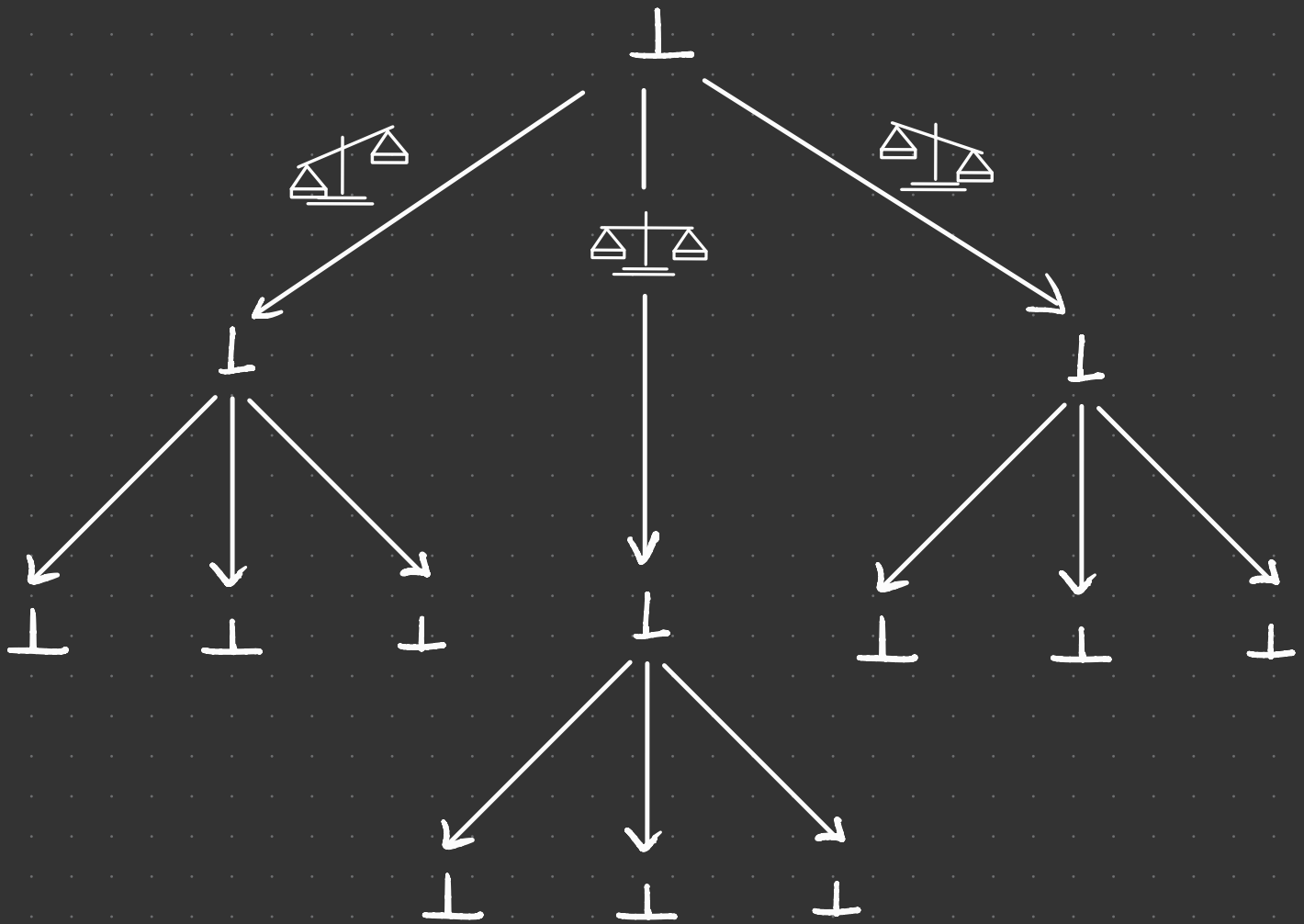
$\Rightarrow$ COULD BE THE START OF AN OPTIMAL STRATEGY!


{+1,+2,+3,+4,
-5,-6,-7,-8,-9}
0,1,2,3,4
5,6,7,8,9
{-1,-2,-3,-4,
+5,-6,+7,+8,+9}
1,2,5
3,4,6
{0,+10,+11,
+12,+13,-10,
-11,-12,-13}
+1,+2,
-6
1
2


{+1,+2,+3,+4,
-5,-6,-7,-8,-9}
0.1.2.3.4
5.6.7.8.9
{-1,-2,-3,-4,
+5,+6,+7,+8,+9}
3.4.5
1.2.6
{+3,+4,-6}
{+1,+2,-5}
{0,+10,+11,
+12,+13,-10,
-11,-12,-13}
{+10,-11,-12}


{+1,+2,+3,+4,
-5,-6,-7,-8,-9}
0,1,2,3,4 ⊥ 5,6,7,8,9
{-1,-2,-3,-4,
+5,-6,+7,+8,+9}
3.4.5 ⊥ 1.2.6
{+3,+4,-6}
{+1,+2,-5}
{0,+10,+11,
+12,+13,-10,
-11,-12,-13}
3.4.5 ⊥ 1.2.6
3 ⊥ 4
⊥
1 ⊥ 2
0.10 ⊥ 11.12
⊥
⊥
⊥
+3
-6
+4
{+10,-11,-12}
11 ⊥ 12
⊥
⊥
-12
+10
-11





EXERCISE

Can we use the 20 questions approach to solve the 14 billiard balls riddle?

EXERCISE

Can we use the 20 questions approach to solve the 14 billiard balls riddle?

SOLUTION

No, because the kind of questions that we can "ask", when we are weighing, is quite limited.

For instance, the first question cannot be "is 1 or 2 heavy?"

Here is a way to formalize the Billiard Balls problem.

Let us identify the solution with the variable X :

"all balls equal"	$X = 0$
"Ball 1 heavy"	$X = 1$
"Ball 1 light"	$X = -1$
"Ball 2 heavy"	$X = 2$
"Ball 2 light"	$X = -2$
$\vdots$	$\vdots$
"Ball 13 heavy"	$X = 13$
"Ball 13 light"	$X = -13$

Hence, $X \in \mathcal{X} = \{ -13, -12, \ldots, -1, 0, 1, \ldots, 12, 13\}$.

Evidently, $|\mathcal{X}| = 27$.

The results of the weighings uniquely specify the value of X.

Hence, in the billiard balls problem, we implicitly specify a ternary code for a certain source.

We know that the number of ternary symbols needed to represent the source is *at least*

$$N \geq H_{D=3}(X).$$

Our code should work *irrespective of the source distribution.* In other words, it must work for all source distributions, thus

$$N \geq \max_{p(x)} H_{D=3}(X) = \log_3 |\mathcal{X}| = \log_3 27 = 3.$$

Hence, conclusion: **We need at least 3 weighings.**

Billiard Balls : Strategies

But is there a strategy that requires only 3 weighings?

From source compression, we can establish the following facts:

- For each weighing, the three outcomes must be *equally likely*.
- The weighings must be independent of each other.

In class, we will together formally prove these two statements.

Then, leveraging these two insights, we will construct the weighing strategy.

Remark: It is because we carefully selected the numbers (alphabet size of 27; each weighing has 3 possible outcomes) that there is a strategy that *exactly* matches the entropy lower bound of 3 weighings. If you change the numbers, it will not generally be true that there is a strategy that *exactly* matches the lower bound.